

Cahier de TPs

Linux - Administration avancée

jour 1

François MICAUX - ACTILIS - 26.10

Table des matières

JOUR 1

Module 1 Installation d'AlmaLinux 10 3

Module 2 Installation automatisée avec Kickstart 6

Module 3 Mettre à jour le système 11

Module 4 Maîtriser la compilation des logiciels 16

Module 1

Installation d'AlmaLinux 10

TP : une installation manuelle, point de départ du kickstart

Objectif et machines

Installer à la main **une première VM AlmaLinux 10**, avec la racine sur LVM. Elle devient, au TP suivant, le **serveur du fichier kickstart** qui installera les autres machines.

Le poste de travail

Le système fourni est un **AlmaLinux 9** avec KVM et `libvirt`. Il héberge les VM et leur réseau `default` (`192.168.122.0/24`, NAT), sur lequel elles se voient entre elles.

La VM `vm1-ks`

2 Go de RAM, 2 cœurs, **un disque virtuel de 10 Go** (`vda`), réseau `default`, accès à Internet.

Partitionnement imposé

Partition	Taille	Type	Format
`/boot`	1 Gio	standard	ext4
`swap`	1 Gio	standard	swap
`/`	5 Gio	LV `lvroot` dans le VG `rootvg`	XFS
PV de `rootvg`	**tout le reste du disque**	partition LVM	

`/boot` reste hors LVM : le chargeur de démarrage doit pouvoir le lire sans passer par LVM (voir slide « Partitionnement : Quelles partitions créer ? »).

Étape 1 : Récupérer l'ISO

Sur le poste de travail, télécharger l'ISO **de démarrage** d'AlmaLinux 10 (`boot`, environ 1 Go), qui prend les paquets sur le réseau pendant l'installation, ainsi que le fichier des empreintes :

```
# cd /var/lib/libvirt/images
U=https://repo.almalinux.org/almalinux/10/isos/x86_64
curl -L -o AlmaLinux-10-boot.iso $U/AlmaLinux-10-latest-x86_64-boot.iso
curl -L -O $U/CHECKSUM
```

Contrôler l'empreinte de l'ISO, en cherchant dans `CHECKSUM` la ligne de `AlmaLinux-10-latest-x86_64-boot.iso` :

```
# grep 'AlmaLinux-10-latest-x86_64-boot.iso)' CHECKSUM
sha256sum AlmaLinux-10-boot.iso
```

Les deux empreintes doivent être **identiques**.

Étape 2 : Choisir l'ISO

> ⚠ Choisir l'ISO `x86_64`
>
> Les ISO AlmaLinux 10 existent aussi en variante `x86_64_v2`, pour les processeurs anciens. Avec elle, un kickstart lancé sous KVM peut rester bloqué au partitionnement (signalé sur le forum AlmaLinux). Le fichier `latest-x86_64` ci-dessus convient.

> ℹ Autres ISO
>
> `minimal` (1,6 Go) contient un système de base, `dvd` (10 Go) tous les paquets. Les deux servent à installer sans réseau : le TP garde `boot`.

Étape 3 : Créer la VM

Créer la VM et lancer l'installateur :

```
# sudo virt-install --name vm1-ks --memory 2048 --vcpus 2 --cpu \
    host-passthrough \
    --disk size=10,bus=virtio --network network=default \
    --cdrom /var/lib/libvirt/images/AlmaLinux-10-boot.iso --os-variant \
    almalinux9
```

La console graphique de la VM s'ouvre toute seule.

> ℹ Valeur de `--os-variant`
>
> La base de données de systèmes (`osinfo`) d'un AlmaLinux 9 ne connaît pas forcément `almalinux10` : `almalinux9` donne les mêmes réglages de VM.

Étape 4 : Installer, écrans de base

1. Choisir **Install AlmaLinux 10** au menu de démarrage, puis la langue **Français** et le clavier **Français (AZERTY)**.
2. **Réseau et nom d'hôte** : activer l'interface, nom d'hôte `srv-ks.exemple.lan`.
3. **Source d'installation** : laisser le **miroir le plus proche**, il est détecté sur le réseau.
4. **Sélection des logiciels** : **Installation minimale**, sans module complémentaire.
5. **Utilisateur** : nom `admin`, mot de passe de son choix, case **Faire de cet utilisateur un administrateur** cochée. **Mot de passe root** : compte désactivé.

Étape 5 : Installer, partitionnement

1. **Destination de l'installation** : sélectionner le disque de 10 Go, choisir **Personnalisée**, puis **Terminé**.
2. Dans l'écran de partitionnement, laisser le schéma sur **LVM**.
3. **+** `/boot`, `1 GiB` : type **Partition standard**, système de fichiers **ext4**, **Mettre à jour les paramètres**.
4. **+** `swap`, `1 GiB` : type **Partition standard**.
5. **+** `/`, `5 GiB` : type **LVM**, système de fichiers **XFS**, nom du volume `lvroot`. Pour « Groupe de volumes », **Modifier…** : nom `rootvg`, politique de taille **Aussi grand que possible**.
6. **Terminé**, relire la liste, **Accepter les modifications**, puis **Commencer l'installation**.

> ⚠ Ordre de création
>
> Créer les partitions dans l'ordre `/boot`, `swap`, `/` : la partition LVM se place alors en dernier, sur tout l'espace restant.

> **BIOS : créer d'abord la partition `biosboot`**
>
> Sur une VM en BIOS, AlmaLinux 10 partitionne en GPT, et GRUB n'a alors aucun endroit où s'installer sans une partition `biosboot` (type **BIOS Boot**, taille minimale, sans système de fichiers).
> La créer **avant** la partition LVM de `/` (**+** `biosboot`) : une fois le groupe de volumes réglé sur **Aussi grand que possible**, il occupe tout l'espace restant et l'installation échoue faute de place.

Après la copie, **redémarrer** la VM (l'ISO n'est plus utilisée) et ouvrir une session avec `admin`.

Étape 6 : Contrôler l'installation

```
$ lsblk
findmnt /boot
ip -br address
```

```
# sudo vgs
sudo lvs
```

Questions :

1. Sur quel disque et quelle partition se trouve le PV de `rootvg` ?
2. Quel espace reste libre dans `rootvg` ?
3. Quelle adresse IP la VM a-t-elle reçue ? La noter, elle sert au TP suivant.

Étape 7 : Lire le fichier kickstart déposé

L'installateur a écrit les réponses données dans `/root/anaconda-ks.cfg`, lisible seulement par `root` :

```
# sudo cat /root/anaconda-ks.cfg
```

Repérer, dans ce fichier :

Ce qui a été choisi	Où le trouver
Source des paquets	ligne `url` ou `cdrom`, selon l'ISO et le choix de la source
Disque et partitions	`ignoredisk`, `part`, `volgroup`, `logvol`
Compte `admin`	ligne `user` (mot de passe chiffré, `--iscrypted`)
Compte root désactivé	ligne `rootpw --lock`
Paquets installés	section `%packages`

Questions :

1. Quelle commande décrit le volume logique `lvroot`, et quelle taille porte-t-elle ?
2. Le mot de passe de `admin` apparaît-il en clair ?

> **À savoir**
>
> Ce fichier reflète **cette** installation, avec le nom de la VM et le disque `vda`. Il sert de point de départ : il se copie, s'édite et se rejoue au TP suivant.

Étape 8 : La complétion de `bash`

Vérifier si l'extension de complétion de `bash` est déjà là, le résultat sert au TP suivant :

```
$ rpm -q bash-completion
```

Module 2

Installation automatisée avec Kickstart

TP : servir un fichier kickstart et installer N machines

Objectif et prérequis

Faire de `vm1-ks` le **serveur de son propre fichier kickstart**, l'améliorer, puis installer plusieurs machines identiques **sans répondre à aucune question**.

Ce qui est déjà en place

La VM `vm1-ks` du TP « Installation d'AlmaLinux 10 », avec son `/root/anaconda-ks.cfg`, et l'ISO `AlmaLinux-10-boot.iso` sur le poste de travail (AlmaLinux 9, KVM).

Les trois améliorations

1. le paquet `bash-completion` installé d'office,
2. un traitement `%post` : `sudo` sans mot de passe pour les administrateurs (`NOPASSWD:`),
3. un traitement `%post` : l'adresse IP affichée dans `/etc/issue`, au-dessus de l'invite de connexion (séquence `\4`).

> ⚠ Les deux machines ne jouent pas le même rôle
>
> Les commandes `virt-install` et `virsh` se tapent sur le **poste de travail**. Tout le reste se tape dans `vm1-ks`, sauf mention contraire. L'adresse de `vm1-ks`, relevée au TP précédent, est notée `IP1` dans les commandes.

Étape 1 : Préparer `/srv/kickstart`

Dans `vm1-ks` :

```
# sudo mkdir -p /srv/kickstart
sudo cp /root/anaconda-ks.cfg /srv/kickstart/ks.cfg
sudo chown -R admin: /srv/kickstart
chmod 644 /srv/kickstart/ks.cfg
```

`anaconda-ks.cfg` n'est lisible que par `root` : la copie reçoit les droits qui permettent au serveur HTTP de la lire.

> ⛔ Le mot de passe chiffré est publié
>
> Le fichier contient le **hachage** du mot de passe de `admin`. Quiconque atteint le serveur HTTP peut le récupérer et chercher à le casser. Le TP se déroule sur un réseau fermé, un déploiement réel utiliserait un hachage dédié aux installations et un réseau restreint.

Étape 2 : Le servir en HTTP

Dans `vm1-ks`, deux commandes :

```
# sudo dnf install -y python3
sudo firewall-cmd --add-port=8080/tcp
```

Lancer le serveur dans un second terminal de la VM, il tourne au premier plan :

```
$ python3 -m http.server 8080 --directory /srv/kickstart
```

Depuis le **poste de travail**, contrôler que le fichier est servi :

```
$ curl -sI http://IP1:8080/ks.cfg | head -1
```

`HTTP/1.0 200 OK` confirme que le fichier est accessible. Une réponse refusée ou muette désigne le pare-feu de `vm1-ks`.

> **La règle `firewall-cmd` est temporaire**
>
> Sans `--permanent`, le port s'ouvre jusqu'au prochain rechargement ou redémarrage du pare-feu. Cela suffit pour ce TP.

Étape 3 : Adapter la source des paquets

Ouvrir `/srv/kickstart/ks.cfg` et regarder la ligne de **source d'installation** :

- une ligne `url --url=...` ou `url --mirrorlist=...` : elle pointe vers le réseau, **la garder**,
- une ligne `cdrom` : elle suppose un média dans la machine installée, la remplacer par les deux dépôts réseau ci-dessous.

```
url --url="https://repo.almalinux.org/almalinux/10/BaseOS/x86_64/os/"
repo --name="AppStream" --
baseurl="https://repo.almalinux.org/almalinux/10/AppStream/x86_64/os/"
```

Contrôler aussi que le disque est bien `vda` (`ignoredisk`, `part`) et que l'installation est en mode `text`, ce qui évite d'ouvrir une console graphique par machine. Une installation faite en mode graphique a écrit `graphical` : le remplacer par `text`.

Étape 4 : Tester le fichier tel quel

Créer une VM `vm2-ks` sur le poste de travail. L'ISO ne sert qu'à lancer l'installateur, les paquets viennent du réseau.

```
# sudo virt-install --name vm2-ks --memory 2048 --vcpus 2 --cpu \
  host-passthrough \
  --disk size=10,bus=virtio --network network=default \
  --cdrom /var/lib/libvirt/images/AlmaLinux-10-boot.iso --os-variant \
  almalinux9
```

Dans la console, au menu de démarrage :

1. sélectionner **Install AlmaLinux 10**,
2. appuyer sur **e** (BIOS) ou **Tab** (UEFI) : la ligne de commande du noyau s'affiche,
3. ajouter à la fin `inst.ks=http://IP1:8080/ks.cfg`, puis **Entrée**.

L'installateur ne pose plus de question, il lit le fichier, partitionne, installe et redémarre.

> **Machine en UEFI**
>
> Sur un menu de type GRUB (UEFI), la touche **e** remplace **Tab** : ajouter `inst.ks=...` à la ligne qui commence par `linuxefi` ou `linux`, puis **Ctrl+x**.

Dans le terminal qui sert HTTP, les requêtes de `vm2-ks` apparaissent : c'est la preuve que le fichier a été lu.

Étape 5 : Améliorer le fichier

Éditer `/srv/kickstart/ks.cfg` dans `vm1-ks`. Dans la section `%packages`, ajouter une ligne :

```
%packages
@^minimal-environment
bash-completion
%end
```

Tout à la fin du fichier, après les autres sections, ajouter :

```
%post --log=/root/ks-post.log
echo "%wheel ALL=(ALL) NOPASSWD: ALL" > /etc/sudoers.d/10-wheel-nopasswd
chmod 0440 /etc/sudoers.d/10-wheel-nopasswd
visudo -cf /etc/sudoers.d/10-wheel-nopasswd
echo 'Adresse IP : \4' >> /etc/issue
%end
```

Étape 6 : Ce que fait le `%post`

Ce que fait chaque ligne

Le fichier de `/etc/sudoers.d/` donne à tout le groupe `wheel` le droit de lancer des commandes sans mot de passe. Il garde les droits `0440` attendus par `sudo`, et `visudo -cf` en contrôle la syntaxe : son verdict (`parsed OK`) se lit dans `/root/ks-post.log`. Les guillemets simples de la dernière ligne gardent `\4` tel quel dans le fichier : c'est `agetty`, au moment d'afficher l'invite, qui le remplace par l'adresse IP de l'interface.

`NOPASSWD` ne convient pas partout

Le droit d'élévation sans mot de passe sert ici pour automatiser et pour le TP. Sur un serveur de production, il augmente les conséquences d'un compte `admin` compromis.

Étape 7 : Installer N machines

Sur le poste de travail, `virt-install` ajoute lui-même `inst.ks` à la ligne de commande du noyau : plus rien à taper dans la console. Trois machines à la suite :

```
# for i in 3 4 5; do
    sudo virt-install --name vm$i-ks --memory 2048 --vcpus 2 --cpu \
    host-passthrough \
      --disk size=10,bus=virtio --network network=default \
      --location /var/lib/libvirt/images/AlmaLinux-10-boot.iso \
    --os-variant almalinux9 \
      --extra-args "inst.ks=http://IP1:8080/ks.cfg" --noautoconsole
done
```

Suivre l'avancement, l'installation prend quelques minutes :

```
$ sudo virsh list --all
```

Chaque installation s'arrête ou redémarre à la fin du fichier. Une machine restée éteinte se relance par `sudo virsh start vm3-ks`.

`--location` ou `--cdrom`

Avec `--location`, `virt-install` extrait lui-même le noyau et l'initrd de l'ISO pour y mettre ses arguments. Avec `--cdrom`, comme à l'étape 4, il faut taper `inst.ks` dans le menu de démarrage.

Étape 8 : Vérifier les machines

Relever l'adresse de chaque machine, puis tester depuis le poste de travail :

```
# sudo virsh domifaddr vm3-ks
```

```
$ ssh admin@IP3 'sudo whoami; rpm -q bash-completion; hostname'
```

Constater :

- `sudo` n'a pas demandé de mot de passe, et répond `root`,
- `bash-completion` est installé, alors qu'il manquait sur `vm1-ks` si le TP précédent l'a constaté,
- l'adresse IP s'affiche dans la console de la VM, au-dessus de l'invite de connexion `login:`,
- `/etc/issue` contient `\4` tel quel : `cat /etc/issue`.

Questions :

1. Quel est le nom d'hôte de chaque machine installée ? Est-ce un souci ?
2. Où trouver la trace de l'exécution du `%post` sur la machine installée ?

Étape 9 : Un fichier par machine (facultatif)

Pour donner son propre nom à chaque machine, générer une copie du fichier par machine, dans `vm1-ks` :

```
$ for i in 6 7; do
    sed "s/--hostname=.*/--hostname=srv-ks$i.exemple.lan/" \
    /srv/kickstart/ks.cfg > /srv/kickstart/ks-$i.cfg
  done
```

Puis installer `vm6-ks` et `vm7-ks` comme à l'étape 7, avec `inst.ks=http://IP1:8080/ks-$i.cfg`. Contrôler le nom d'hôte de chacune avec `hostname`.

Étape 10 : Un conteneur `httpd` à la place de Python (facultatif)

Dans `vm1-ks`, arrêter le serveur Python (**Ctrl+c**), puis :

```
$ sudo dnf install -y podman
  podman run -d --name ks-httpd -p 8080:80 -v \
    /srv/kickstart:/usr/local/apache2/htdocs:ro,Z \
    docker.io/library/httpd:2.4
  curl -sI http://localhost:8080/ks.cfg | head -1
```

Le même `inst.ks=http://IP1:8080/ks.cfg` fonctionne. L'option `:Z` donne aux fichiers le contexte SELinux que le conteneur peut lire.

> **À savoir**
>
> `/usr/local/apache2/htdocs` est le répertoire servi par l'image officielle `httpd`. Un conteneur lancé par `admin` n'a pas besoin de `sudo`, et le port 8080 dépasse la limite des ports privilégiés.

Étape 11 : Un ISO qui démarre sur le kickstart (facultatif)

Le fichier reste sur `vm1-ks`, servi en HTTP. Sur le poste de travail, `mkksiso` ajoute `inst.ks=` aux lignes de démarrage d'une copie de l'ISO :

```
# dnf install -y lorax
  D=/var/lib/libvirt/images
  mkksiso --cmdline "inst.ks=http://IP1:8080/ks.cfg" \
    $D/AlmaLinux-10-boot.iso $D/AlmaLinux-10-ks.iso
```

Installer `vm8-ks` à partir de ce nouvel ISO :

```
# sudo virt-install --name vm8-ks --memory 2048 --vcpus 2 --cpu \
    host-passthrough \
    --disk size=10,bus=virtio --network network=default \
    --cdrom /var/lib/libvirt/images/AlmaLinux-10-ks.iso --os-variant \
    almalinux9
```

Au menu de démarrage, choisir **Install AlmaLinux 10** : la ligne de commande contient déjà `inst.ks=`, rien n'est à compléter.

> **Le fichier est lu à l'installation**
>
> L'ISO ne contient que l'adresse : modifier `ks.cfg` sur `vm1-ks` suffit, sans refaire l'ISO. Le serveur HTTP de `vm1-ks` doit répondre pendant l'installation.

> **Toutes les entrées du menu**
>
> `mkksiso` ajoute l'argument à **toutes** les entrées de démarrage de l'ISO (BIOS et UEFI), il ne crée pas d'entrée séparée « Installation KS ». Une entrée supplémentaire demande de modifier à la main `isolinux.cfg` et `grub.cfg` dans l'ISO.

Étape 12 : Nettoyer

Sur le poste de travail, supprimer les VM de test et leurs disques, en gardant `vm1-ks` et `vm3-ks` (source du TP « Booter autrement ») :

```
# for v in vm2-ks vm4-ks vm5-ks vm6-ks vm7-ks vm8-ks; do
    sudo virsh destroy $v 2>/dev/null
    sudo virsh undefine $v --remove-all-storage 2>/dev/null
done
```

Supprimer aussi `/var/lib/libvirt/images/AlmaLinux-10-ks.iso`. Dans `vm1-ks`, arrêter le service de fichiers : **Ctrl+c** pour Python, `podman rm -f ks-httpd` pour le conteneur.

Module 3

Mettre à jour le système

TP : rpm, dépôts, recherche, historique et mises à jour avec dnf

Objectif et machine

Interroger la base **RPM**, ouvrir un paquet **sans l'installer**, lire la configuration de **DNF**, puis chercher, installer, annuler et **mettre à jour** avec `dnf`.

La machine

Une VM AlmaLinux 10 avec accès à Internet et un compte qui dispose de `sudo`. Tout se fait en SSH.

```
$ ssh admin@vm3-ks
```

> ⚠ Premier appel de `dnf`
>
> Le premier appel télécharge les méta-données des dépôts : plusieurs minutes selon la connexion. Les suivants utilisent le cache.

Étape 1 : Interroger la base RPM

```
$ rpm -q bash
rpm -qi bash | head -8
rpm -ql bash | head -3
rpm -qc bash
```

Questions :

1. Comment lister seulement les fichiers de configuration d'un paquet ?

```
$ rpm -qf /etc/passwd
```

Questions :

2. Quel paquet a installé `/etc/passwd` ?

```
$ rpm -q --requires bash | head -3
rpm -q --scripts systemd | head -4
```

Questions :

3. Que contient la section `preinstall scriptlet` de `systemd` ?

Étape 2 : Ouvrir un paquet sans l'installer

Télécharger le paquet `tree` sans l'installer, puis l'examiner :

```
$ mkdir ~/tp-rpm && cd ~/tp-rpm
dnf download tree
rpm -qpi tree-*.rpm | head -3
rpm -qpl tree-*.rpm
rpm --checksig tree-*.rpm
```

Extraire son contenu avec `rpm2cpio` :

```
$ rpm2cpio tree-*.rpm | cpio -idm
find . -type f
```

Questions :

1. Que signifie `digests signatures OK` ?
2. Où `tree` serait-il installé, et où est-il extrait ?

Étape 3 : Configuration et dépôts

```
$ ls -l /etc/yum.conf
cat /etc/dnf/dnf.conf
```

Questions :

1. Vers quoi pointe `/etc/yum.conf` ?

```
$ ls /etc/yum.repos.d
dnf repolist
dnf repolist all | head -12
```

Questions :

2. Quels dépôts sont actifs ? Le dépôt `crb` l'est-il ?

```
$ dnf repoinfo baseos | head -14
```

Questions :

3. Quelle durée de vie des méta-données le dépôt `baseos` annonce-t-il (`Repo-expire`) ?

Étape 4 : Chercher

```
$ dnf provides '*/lspci'
```

Questions :

1. Quel paquet fournit `lspci` ?
2. Que signifie `Repo : @System` dans la réponse de `dnf provides` ?

```
$ dnf info tree | head -8
dnf search tree | head -5
dnf list --showduplicates tree
dnf repoquery --requires tree
```

Questions :

3. Dans quel dépôt se trouve `tree` ?

Étape 5 : Installer, annuler

Installer le RPM téléchargé à l'étape 2, qui passe par `dnf` pour les dépendances :

```
$ sudo dnf -y install ./tree-*.rpm
tree --version
sudo dnf history | head -4
sudo dnf history info last | head -14
```

Annuler la transaction, puis installer depuis le dépôt :

```
$ sudo dnf -y history undo last
rpm -q tree
sudo dnf -y install tree
sudo dnf -y remove tree
```

Questions :

1. Quelle ligne de `dnf history info` indique d'où vient le paquet installé ?
2. Que fait `history undo last` d'une transaction d'installation ?

Étape 6 : Ajouter un dépôt : installer `micro`

`micro` est un éditeur de texte en terminal. Il ne figure dans aucun dépôt d'AlmaLinux, mais dans **EPEL** (*Extra Packages for Enterprise Linux*), un dépôt communautaire du projet Fedora :

```
$ dnf info micro
```

Questions :

1. Que répond `dnf info micro` ?

Le paquet `epel-release` (dans `extras`) dépose le fichier `.repo` d'EPEL et sa clé de signature :

```
# sudo dnf install epel-release
rpm -ql epel-release
dnf repolist
```

Questions :

2. Quels fichiers `epel-release` installe-t-il pour déclarer le dépôt et vérifier ses paquets ?
3. Quel dépôt est apparu dans `dnf repolist` ?

```
$ dnf info micro | head -8
```

```
# sudo dnf install micro
```

La clé d'EPEL est proposée à l'import au premier paquet installé depuis ce dépôt.

Questions :

4. Dans quel dépôt `dnf info` trouve-t-il `micro` ?
5. Que demande `dnf` avant d'installer, et pourquoi ?

```
$ micro -version
rpm -qf /usr/bin/micro
dnf repoquery --installed --qf '%{name} %{from_repo}' micro
```

Ouvrir un fichier avec `micro` (Ctrl-S enregistre, Ctrl-Q quitte) :

```
$ micro ~/essai.txt
```

Questions :

6. Quelle version de `micro` est installée, et de quel dépôt provient-elle ?

> ⚠ Dépôts tiers
>
> EPEL n'est pas maintenu par AlmaLinux et ne bénéficie pas de son support. Ses paquets suivent le rythme du projet Fedora, avec leurs mises à jour. Chaque dépôt ajouté s'ajoute à ce qu'il faut surveiller en production.

Retirer ensuite `micro`, puis désactiver le dépôt sans le supprimer :

```
# sudo dnf remove micro
sudo dnf config-manager --set-disabled epel
dnf repolist
```

Questions :

7. Quelle commande réactive EPEL pour une seule commande, sans modifier le fichier `.repo` ?

Étape 7 : Repérer les mises à jour

```
$ dnf check-update | tail -3 ; echo "code retour : ${PIPESTATUS[0]}"
dnf updateinfo list | head -5
dnf list updates | head -4
sudo dnf upgrade --security --assumeno | head -12
```

`--assumeno` répond « non » à la confirmation : rien n'est installé.

Questions :

1. Quel code retour `check-update` donne-t-il quand des mises à jour sont disponibles ?
2. Que signifie `ALSA-2026:...` dans `updateinfo list` ?
3. Quelle est la différence entre `dnf upgrade` et `dnf upgrade --security` ?

> **À savoir**
>
> `ALSA` signifie *AlmaLinux Security Advisory*. Une mise à jour du noyau s'**installe** à côté du noyau en cours (« Installing: kernel ») : le nouveau noyau ne tourne qu'après un redémarrage.

Étape 8 : Modules et groupes

```
$ dnf module list
```

Questions :

1. Que renvoie `dnf module list` sur AlmaLinux 10 ?

```
$ dnf grouplist | head -8
dnf groupinfo "Development Tools" | head -12
```

Questions :

2. Quels paquets `groupinfo` classe-t-il en « Mandatory » ?

Étape 9 : Le cache

```
$ du -sh /var/cache/dnf
sudo dnf clean expire-cache
```

Questions :

1. Que fait `dnf clean expire-cache` : supprime-t-il les méta-données ?

```
$ dnf repolist | head -2
grep -h metadata_expire /etc/yum.repos.d/almalinux-baseos.repo
```

Questions :

2. Pourquoi éviter `dnf clean all` en réflexe ?

Nettoyer

`rm -r ~/tp-rpm` supprime le dossier de travail de l'étape 2.

Module 4

Maîtriser la compilation des logiciels

TP : compiler GNU Hello, écrire un Makefile, publier une bibliothèque partagée

Objectif et machine

Compiler un logiciel **depuis son archive de sources**, l'installer dans `/opt`, écrire un **Makefile** à la main, puis **publier une bibliothèque partagée** pour que le chargeur la trouve.

La machine

La VM `vm3-ks` (ou toute VM AlmaLinux 10 reliée à Internet), avec un compte qui dispose de `sudo`. Tout se fait en SSH.

```
$ ssh admin@vm3-ks
```

> **Le logiciel**
>
> **GNU Hello** : un programme de quelques lignes qui affiche un message, mais dont l'archive suit toute la démarche GNU (`configure`, `make`, tests, pages de manuel, traductions). Il n'a aucune dépendance au-delà du compilateur.

Étape 1 : Préparer la machine

Le compilateur et `make` viennent du groupe `Development Tools`. Le paquet `man-db` sert à l'étape 4.

```
$ sudo dnf group install "Development Tools"
sudo dnf install man-db
```

Vérifier :

```
$ gcc --version | head -1
make --version | head -1
```

Questions :

1. Quelle commande installe d'un coup `gcc`, `make`, `autoconf` et `patch` ?
2. Que signale un message `configure: error: ... not found` et comment retrouver le paquet à installer ?

Étape 2 : Récupérer et vérifier l'archive

Travailler dans un répertoire de sources, avec un **utilisateur ordinaire** :

```
$ mkdir -p ~/src && cd ~/src
curl -O https://mirrors.kernel.org/gnu/hello/hello-2.12.3.tar.gz
```

Contrôler la somme de l'archive. Elle a été relevée sur le miroir le 2026-10-06. Créer le fichier `hello.sha256` avec un éditeur, contenant la somme, deux espaces, puis le nom de l'archive :

```
≡ 0d5f60154382fee10b114a1c34e785d8b1f492073ae2d3a6f7b147687b366aa0  hello-2.12.3.tar.gz
```

Puis :

```
$ sha256sum -c hello.sha256
```

Étape 3 : Extraire et lire

Lister le contenu **avant** de l'extraire, puis extraire et lire la documentation d'accueil :

```
$ tar tf hello-2.12.3.tar.gz | head
tar xf hello-2.12.3.tar.gz
cd hello-2.12.3
less README INSTALL
```

Questions :

1. Que répond `sha256sum -c` si l'archive est intacte ?
2. Pourquoi lister le contenu de l'archive avant de l'extraire ?

> **À savoir**
>
> Le projet GNU publie aussi une signature (`.sig`) de chaque archive, vérifiable avec `gpg`. C'est un contrôle plus fort qu'une somme relevée sur le même serveur que l'archive.

Étape 4 : Configurer, compiler, tester

Lister les options de `configure` qui concernent l'emplacement de l'installation :

```
$ ./configure --help | grep -E -- '^ +--(prefix|sysconfdir|mandir)='
```

Configurer pour une installation dans `/opt/hello`, puis compiler **en parallèle** :

```
$ ./configure --prefix=/opt/hello
```

Questions :

1. Quelle option de `configure` fixe le répertoire d'installation ?

```
$ make -j $(nproc)
```

Questions :

2. Quel utilisateur a lancé `configure`, `make` et `make check` ?

```
$ make check
```

Questions :

3. Combien de tests ont réussi, combien ont échoué ?

Étape 5 : Installer et finaliser

Seule cette étape demande `root` :

```
$ sudo make install
```

Observer ce qui a été installé et par qui :

```
$ find /opt/hello -maxdepth 3 | sort
ls -l /opt/hello/bin
/opt/hello/bin/hello
```

Étape 6 : Finaliser

`hello` n'est pas encore dans le `PATH`, et `man hello` ne trouve rien. **Finaliser à la main**, comme le dit la slide « Compilation, tests, installation » :

Créer `/etc/profile.d/hello.sh` avec un éditeur lancé en root (`sudo`) :

```
export PATH=/opt/hello/bin:$PATH
```

Ajouter une ligne à la fin de `/etc/man_db.conf` avec un éditeur lancé en root (`sudo`) :

```
MANDATORY_MANPATH /opt/hello/share/man
```

Puis :

```
$ source /etc/profile.d/hello.sh
command -v hello
man hello | head -5
```

Questions :

1. Quels sont les propriétaires des fichiers de `/opt/hello` ?
2. Quels fichiers modifie-t-on pour que le `PATH` et `man` connaissent le logiciel ?
3. Que faudrait-il faire en plus si `hello` installait des bibliothèques partagées ?

> **À conserver**
>
> Le répertoire `~/src/hello-2.12.3` et l'installation dans `/opt/hello` sont à garder sur la VM.

Étape 7 : Un Makefile à la main

Dans un nouveau répertoire, trois fichiers : un programme, une fonction, son en-tête.

```
$ mkdir -p ~/src/salut && cd ~/src/salut
```

```
/* salut.h */
void salut(const char *nom);
```

```
/* salut.c */
#include <stdio.h>
#include "salut.h"

void salut(const char *nom)
{
    printf("Bonjour, %s !\n", nom);
}
```

```
/* main.c */
#include "salut.h"

int main(void)
{
    salut("stagiaire");
    return 0;
}
```

Étape 8 : Le Makefile

À taper avec un éditeur. Les lignes de commande commencent par **une tabulation**, pas par des espaces :

```
CC     = gcc
CFLAGS = -O2 -Wall
EXEC   = monprog
OBJS   = main.o salut.o

all: $(EXEC)

$(EXEC): $(OBJS)
        $(CC) $(LDFLAGS) -o $@ $^

%.o: %.c
        $(CC) $(CFLAGS) -c $< -o $@

main.o salut.o: salut.h

clean:
        @rm -f $(OBJS) $(EXEC)
```

> **À savoir**
>
> Un copier-coller depuis un PDF remplace les tabulations par des espaces. `make` répond alors `*** missing separator. Arrêt.` : la ligne désignée doit recommencer par une tabulation.

Étape 9 : Observer les reconstructions

```
$ make
./monprog
make
```

Questions :

1. Que répond le deuxième `make`, lancé sans rien avoir modifié ?
2. Quelle est la première cible, et que lance `make` sans argument ?

```
$ touch salut.c
make
```

Questions :

3. Quelle(s) commande(s) `gcc` s'exécutent après `touch salut.c` ?

```
$ touch salut.h
make
make clean
```

Questions :

4. Et après `touch salut.h` ? Quelle ligne du Makefile l'explique ?

Étape 10 : Une bibliothèque hors des répertoires standard

Construire `salut.c` en **bibliothèque partagée**, l'installer dans `/opt/salut/lib`, puis lier le programme

dessus :

```
$ make clean
gcc -fPIC -shared -o libsalut.so salut.c
sudo install -D libsalut.so /opt/salut/lib/libsalut.so
gcc -O2 -Wall -c main.c -o main.o
gcc -o monprog-dyn main.o -L/opt/salut/lib -lsalut
./monprog-dyn
```

Le programme ne démarre pas. Chercher pourquoi :

```
$ objdump -p monprog-dyn | grep NEEDED
ldd ./monprog-dyn
```

Questions :

1. Quel message d'erreur le chargeur affiche-t-il ?
2. Quelle bibliothèque `ldd` marque-t-il `not found` ?
3. Pourquoi `-L/opt/salut/lib` a-t-il suffi à la compilation et pas à l'exécution ?

Étape 11 : Rendre la bibliothèque visible

Essai **temporaire**, pour ce seul lancement :

```
$ LD_LIBRARY_PATH=/opt/salut/lib ./monprog-dyn
```

Solution **durable** : déclarer le répertoire, puis reconstruire le cache.

Créer `/etc/ld.so.conf.d/salut.conf` avec un éditeur lancé en root (*`sudo`*), contenant le répertoire :

```
≡ /opt/salut/lib
```

Puis reconstruire le cache :

```
$ sudo ldconfig
```

```
$ ldconfig -p | grep salut
ldd ./monprog-dyn
./monprog-dyn
```

Questions :

1. Quelle commande reconstruit le cache lu par le chargeur, et dans quel fichier est-il ?
2. Que se passe-t-il si on oublie de la relancer après avoir créé `salut.conf` ?

Étape 12 : `RPATH` et `RUNPATH` *(facultative)*

Intégrer le chemin au binaire, sans toucher à la configuration du système :

```
$ gcc -o monprog-rp main.o -L/opt/salut/lib -lsalut \
  -Wl,-rpath,/opt/salut/lib
readelf -d monprog-rp | grep -E 'RPATH|RUNPATH'
gcc -o monprog-rp2 main.o -L/opt/salut/lib -lsalut \
  -Wl,--enable-new-dtags,-rpath,/opt/salut/lib
readelf -d monprog-rp2 | grep -E 'RPATH|RUNPATH'
```

Questions :

1. Quel attribut ELF `readelf` montre-t-il pour chacun des deux binaires ?
2. Lequel est consulté avant `LD_LIBRARY_PATH` ?

Étape 13 : `LD_PRELOAD` *(facultative)*

Détourner une fonction de la bibliothèque :

```c
/* hook.c */
#include <stdio.h>

void salut(const char *nom)
{
    printf("Salut détourné pour %s\n", nom);
}
```

```
$ gcc -fPIC -shared -o libhook.so hook.c
LD_PRELOAD=./libhook.so ./monprog-dyn
./monprog-dyn
```

Questions :

1. Quelle fonction s'exécute avec `LD_PRELOAD`, puis sans la variable ?

> **À savoir**
>
> Un binaire où `salut.o` est lié directement, comme `monprog` de l'étape 7 (à reconstruire avec `make`), n'est pas affecté : `LD_PRELOAD=./libhook.so ./monprog` affiche le message d'origine.